1. **Divide:** Break problem into smaller sub-problems.
2. **Conquer:** Recursively solve smaller sub-problems untill base-cases are reached, which are easy to solve.
3. **Combine:** Combine the solutions of the smaller problems into the solution of the current problem.

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

MERGE(A, p, q, r)

```
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = −∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = −∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])            // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

```
ALGORITHM  EfficientClosestPair(P, Q)
    //Solves the closest-pair problem by divide-and-conquer
    //Input: An array P of n ≥ 2 points in the Cartesian plane sorted in
    //        nondecreasing order of their x coordinates and an array Q of the
    //        same points sorted in nondecreasing order of the y coordinates
    //Output: Euclidean distance between the closest pair of points
    if n ≤ 3
        return the minimal distance found by the brute-force algorithm
    else
        copy the first ⌈n/2⌉ points of P to array P_l
        copy the same ⌈n/2⌉ points from Q to array Q_l
        copy the remaining ⌊n/2⌋ points of P to array P_r
        copy the same ⌊n/2⌋ points from Q to array Q_r
        d_l ← EfficientClosestPair(P_l, Q_l)
        d_r ← EfficientClosestPair(P_r, Q_r)
        d ←min{d_l, d_r}
        m ← P[⌈n/2⌉ − 1].x
        copy all the points of Q for which |x − m| < d into array S[0..num − 1]
        dminsq ← d^2
        for i ← 0 to num − 2 do
            k ← i + 1
            while k ≤ num − 1 and (S[k].y − S[i].y)^2 < dminsq
                dminsq ← min((S[k].x − S[i].x)^2+ (S[k].y − S[i].y)^2, dminsq)
                k ← k + 1
    return sqrt(dminsq)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

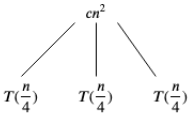
```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

Recursion Trees

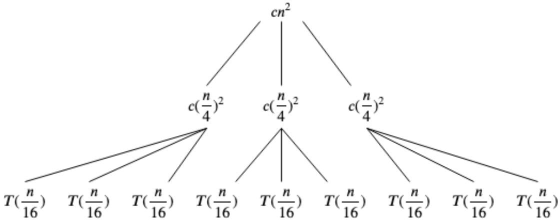
- Gives you an idea of how a recurrence relation will *expand*.
- $T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + \Theta(n^2)$

T(n)

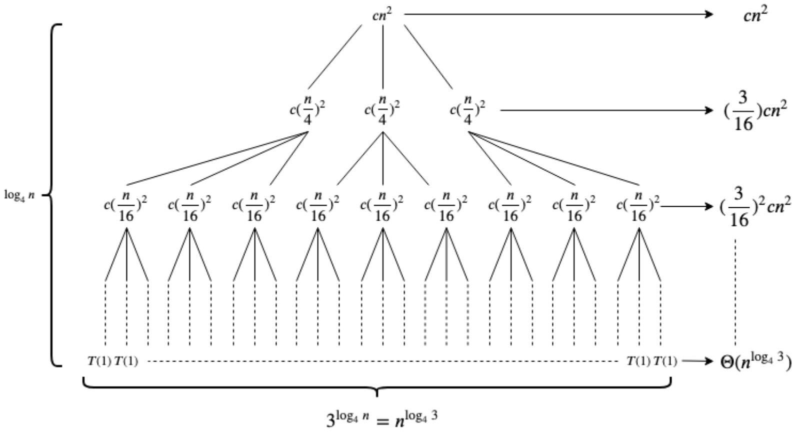
$$T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + cn^2$$



$$T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + cn^2$$



$$T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + cn^2$$



$$T(n) = \sum_{i=0}^{\log_4 n - 1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) = \sum_{i=0}^{\log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) = \sum_{i=0}^{\log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$\leq \sum_{i=0}^{\infty}(\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3})$$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&\le \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&\le \frac{1}{1-\frac{3}{16}} cn^2 + \Theta(n^{\log_4 3})
\end{aligned}
$$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&\leq \sum_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&\leq \frac{1}{1-\frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})
\end{aligned}
$$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&\leq \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&\leq \frac{1}{1-\frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= \mathcal{O}(n^2)
\end{aligned}
$$

I guess!!

- Now, we prove it by using the substitution method.

- Now, we prove it by using the substitution method.
- Our goal is to show:

$$T(n) \leq dn^2$$

for some constant $d > 0$ (which we get to pick).

- Now, we prove it by using the substitution method.
- Our goal is to show:

$$T(n) \leq dn^2$$

  for some constant $d > 0$ (which we get to pick).
- Inductive Hypothesis :

$$T(m) \leq dm^2 \qquad \forall m < n$$

- Now, we prove it by using the substitution method.
- Our goal is to show:

$$T(n) \le dn^2$$

for some constant $d > 0$ (which we get to pick).

- Inductive Hypothesis :

$$T(m) \le dm^2 \qquad\qquad \forall m < n$$

- which gives us:

$$\begin{aligned}
T(n) &\le 3T(\lfloor \frac{n}{4} \rfloor) + cn^2 \\
&\le 3d\lfloor \frac{n}{4} \rfloor^2 + cn^2 \\
&\le 3d(\frac{n}{4})^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\le dn^2 && \text{if we pick } d \ge \frac{16}{13}c
\end{aligned}$$

- Now, we prove it by using the substitution method.
- Our goal is to show:

$$T(n) \leq dn^2$$

  for some constant $d > 0$ (which we get to pick).
- Inductive Hypothesis :

$$T(m) \leq dm^2 \qquad \forall m < n$$

- which gives us:

$$\begin{aligned}
T(n) &\leq 3T(\lfloor \frac{n}{4} \rfloor) + cn^2 \\
&\leq 3d\lfloor \frac{n}{4} \rfloor^2 + cn^2 \\
&\leq 3d(\frac{n}{4})^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\leq dn^2 && \text{if we pick } d \geq \frac{16}{13}c
\end{aligned}$$

- Which completes the proof. Then, $T(n) = \mathcal{O}(n^2)$.

- Did we forget the base case of the induction!?

$$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + \mathcal{O}(n)$$

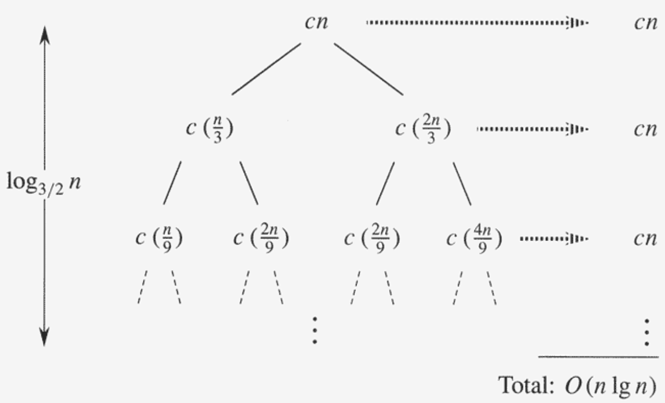


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

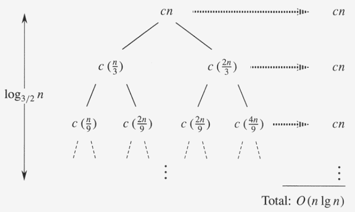


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

- *If*, the tree were complete, there would be

$$2^{\log_{3/2} n} = n^{\log_{3/2} 2}$$

leaves.

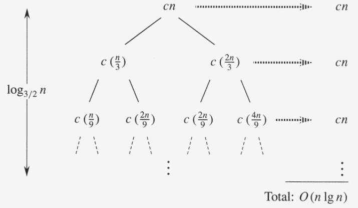


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

- *If*, the tree were complete, there would be

$$2^{\log_{3/2} n} = n^{\log_{3/2} 2}$$

  leaves.
- Is the tree complete?

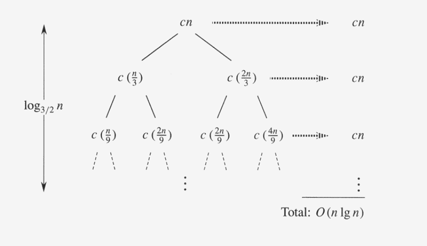


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

- *If*, the tree were complete, there would be

$$2^{\log_{3/2} n} = n^{\log_{3/2} 2}$$

  leaves.
- Is the tree complete?
- Worst case: tree would be complete.

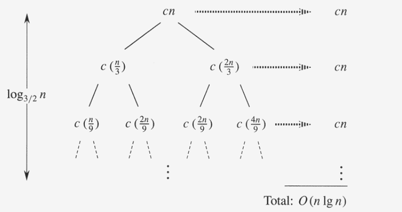


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

- *If*, the tree were complete, there would be

$$2^{\log_{3/2} n} = n^{\log_{3/2} 2}$$

  leaves.
- Is the tree complete?
- Worst case: tree would be complete.
- Best case: tree very degenerate.

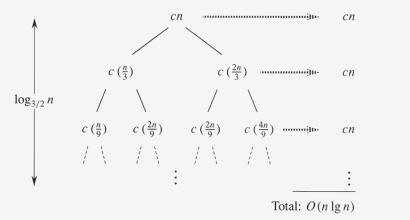


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Let's pretend the tree is complete...

- Each leaf has constant cost.

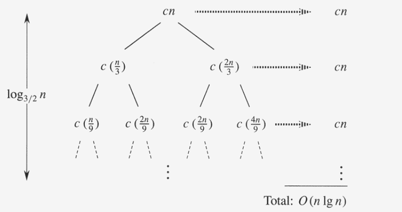


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Let's pretend the tree is complete...

- Each leaf has constant cost.
- $n^{\log_{3/2} 2}$ leaves.

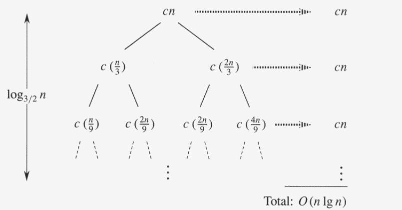


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Let's pretend the tree is complete...

- Each leaf has constant cost.
- $n^{\log_{3/2} 2}$ leaves.
- Cost of all leaves should be $\Theta(n^{\log_{3/2} 2})$. can $T(n)$ possibly be $\mathcal{O}(n \lg n)$?

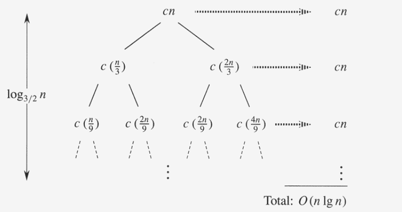


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Let's pretend the tree is complete...

- Each leaf has constant cost.
- $n^{\log_{3/2} 2}$ leaves.
- Cost of all leaves should be $\Theta(n^{\log_{3/2} 2})$. can $T(n)$ possibly be $\mathcal{O}(n \lg n)$?
- Recursion tree is not always accurate.

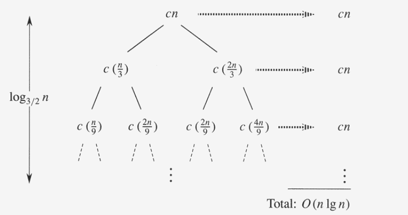


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Let's pretend the tree is complete...

- Each leaf has constant cost.
- $n^{\log_{3/2} 2}$ leaves.
- Cost of all leaves should be $\Theta(n^{\log_{3/2} 2})$. can $T(n)$ possibly be $\mathcal{O}(n \lg n)$?
- Recursion tree is not always accurate.
- Actually, we will guess $T(n) = \mathcal{O}(n \lg n)$.

$$T(n) = T(n/3) + T(2n/3) + cn$$

Our goal: $T(n) \leq dn \lg n$

- Inductive hypothesis :

$$T(m) \leq dm \lg m \qquad\qquad m < n$$

$$T(n) = T(n/3) + T(2n/3) + cn$$

Our goal: $T(n) \leq dn \lg n$

- Inductive hypothesis :

$$T(m) \leq dm \lg m \qquad\qquad m < n$$

- Then,

$$T(n) \leq T(n/3) + T(2n/3) + cn$$

$$T(n) \leq T(n/3) + T(2n/3) + cn$$

$$T(n) \leq d\frac{n}{3}\lg(\frac{n}{3}) + d\frac{2n}{3}\lg(\frac{2n}{3}) + cn$$

$$T(n) \leq T(n/3) + T(2n/3) + cn$$

$$T(n) \leq d\frac{n}{3}\lg(\frac{n}{3}) + d\frac{2n}{3}\lg(\frac{2n}{3}) + cn$$

$$\begin{aligned} T(n) \leq {} & (d\frac{n}{3}\lg n - d\frac{n}{3}\lg 3) \\ & + (d\frac{2n}{3}\lg n - d\frac{2n}{3}\lg(\frac{3}{2})) \end{aligned}$$

$$
\begin{aligned}
T(n) &\leq T(n/3) + T(2n/3) + cn \\
T(n) &\leq d\frac{n}{3}\lg(\frac{n}{3}) + d\frac{2n}{3}\lg(\frac{2n}{3}) + cn \\
T(n) &\leq (d\frac{n}{3}\lg n - d\frac{n}{3}\lg 3) \\
&\quad + (d\frac{2n}{3}\lg n - d\frac{2n}{3}\lg(\frac{3}{2})) \\
T(n) &\leq dn\lg n - d((\frac{n}{3})\lg 3 \\
&\quad + (\frac{2n}{3})\lg(\frac{3}{2})) + cn
\end{aligned}
$$

$$
\begin{aligned}
T(n) &\le T(n/3) + T(2n/3) + cn \\
T(n) &\le d\frac{n}{3}\lg(\frac{n}{3}) + d\frac{2n}{3}\lg(\frac{2n}{3}) + cn \\
T(n) &\le (d\frac{n}{3}\lg n - d\frac{n}{3}\lg 3) \\
&\quad + (d\frac{2n}{3}\lg n - d\frac{2n}{3}\lg(\frac{3}{2})) \\
T(n) &\le dn\lg n - d((\frac{n}{3})\lg 3 \\
&\quad + (\frac{2n}{3})\lg(\frac{3}{2})) + cn \\
T(n) &\le dn\lg n - d((\frac{n}{3}\lg 3 + (\frac{2n}{3})\lg 3 \\
&\quad - (\frac{2n}{3})\lg 2)) + cn
\end{aligned}
$$

$$
\begin{aligned}
T(n) &\leq T(n/3) + T(2n/3) + cn \\
T(n) &\leq d\frac{n}{3}\lg(\frac{n}{3}) + d\frac{2n}{3}\lg(\frac{2n}{3}) + cn \\
T(n) &\leq (d\frac{n}{3}\lg n - d\frac{n}{3}\lg 3) \\
&\quad + (d\frac{2n}{3}\lg n - d\frac{2n}{3}\lg(\frac{3}{2})) \\
T(n) &\leq dn\lg n - d((\frac{n}{3})\lg 3 \\
&\quad + (\frac{2n}{3})\lg(\frac{3}{2})) + cn \\
T(n) &\leq dn\lg n - d((\frac{n}{3}\lg 3 + (\frac{2n}{3})\lg 3 \\
&\quad - (\frac{2n}{3})\lg 2)) + cn \\
T(n) &\leq dn\lg n - dn(\lg 3 - \frac{2}{3}) + cn \\
&\leq dn\lg n \qquad\qquad \text{if } d \geq \frac{c}{\lg 3 - (2/3)}
\end{aligned}
$$

Master Theorem

- Consider the following recurrance relationship:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

Master Theorem

- Consider the following recurrance relationship:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- This may describe a divide and conquer algorithm which breaks a problem of size n into:

Master Theorem

- Consider the following recurrance relationship:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- This may describe a divide and conquer algorithm which breaks a problem of size n into:
  - A number a of subproblems.

Master Theorem

- Consider the following recurrance relationship:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- This may describe a divide and conquer algorithm which breaks a problem of size n into:
  - A number a of subproblems.
  - Each has size $\frac{n}{b}$.

Master Theorem

- Consider the following recurrance relationship:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- This may describe a divide and conquer algorithm which breaks a problem of size n into:
  - A number a of subproblems.
  - Each has size $\frac{n}{b}$.
  - $f(n)$ is the extra cost of dividing, and the cost of combining solutions.

Master Theorem

- Consider the following recurrance relationship:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- This may describe a divide and conquer algorithm which breaks a problem of size n into:
  - A number a of subproblems.
  - Each has size $\frac{n}{b}$.
  - $f(n)$ is the extra cost of dividing, and the cost of combining solutions.
- If we make $a = 2$, $b = 2$, $f(n) = \Theta(n)$, we get a familiar recurrance.

Master Theorem

Theorem

Let $a \geq 1$, $b > 1$ *be constants, let* $f(n)$ *be a function and let* $T(n)$ *be the recurrence:*

$$T(n) = aT(n/b) + f(n)$$

Then, $T(n)$ *has the following asymptotic bounds:*

Master Theorem

Theorem

Let $a \geq 1$, $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be the recurrence:

$$T(n) = aT(n/b) + f(n)$$

Then, $T(n)$ has the following asymptotic bounds:

1. *If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$*

Master Theorem

Theorem

Let $a \geq 1$, $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be the recurrence:

$$T(n) = aT(n/b) + f(n)$$

Then, $T(n)$ has the following asymptotic bounds:

1. *If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$*
2. *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$*

Master Theorem

Theorem

Let $a \geq 1$, $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be the recurrence:

$$T(n) = aT(n/b) + f(n)$$

Then, $T(n)$ has the following asymptotic bounds:

1. *If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$*
2. *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$*
3. *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then, $T(n) = \Theta(f(n))$.*

- How to use the master method?

- How to use the master method?
- Identify a, b, $f(n)$ and compare $f(n)$ with $n^{\log_b a}$.

- How to use the master method?
- Identify a, b, $f(n)$ and compare $f(n)$ with $n^{\log_b a}$.
- Examples:

- $T(n) = 9T(n/3) + n$

- $T(n) = 9T(n/3) + n$
- $a = 9$, $b = 3$, $f(n) = n$.

- $T(n) = 9T(n/3) + n$
- $a = 9,\ b = 3,\ f(n) = n.$
- $n^{\log_3 9} = n^2.$

- $T(n) = 9T(n/3) + n$
- $a = 9,\ b = 3,\ f(n) = n$.
- $n^{\log_3 9} = n^2$.
- $f(n)$ grows slower than n^2! Not only is $f(n) = \mathcal{O}(n^2)$...

- $T(n) = 9T(n/3) + n$
- $a = 9,\ b = 3,\ f(n) = n$.
- $n^{\log_3 9} = n^2$.
- $f(n)$ grows slower than n^2! Not only is $f(n) = \mathcal{O}(n^2)$...
- If you subtract some $\epsilon > 0$ from the exponent of n^2, $f(n)$ would *still* be growing slower.

- $T(n) = 9T(n/3) + n$
- $a = 9,\ b = 3,\ f(n) = n$.
- $n^{\log_3 9} = n^2$.
- $f(n)$ grows slower than n^2! Not only is $f(n) = \mathcal{O}(n^2)$...
- If you subtract some $\epsilon > 0$ from the exponent of n^2, $f(n)$ would *still* be growing slower.
- Since $f(n) = \mathcal{O}(n^{2-1})$, by case 1 of the master method with $\epsilon = 1,\ T(n) = \Theta(n^2)$.

- $T(n) = T(2n/3) + 1$

- $T(n) = T(2n/3) + 1$
- $a = 1$, $b = 3/2$, $f(n) = 1$.

- $T(n) = T(2n/3) + 1$
- $a = 1$, $b = 3/2$, $f(n) = 1$.
- $n^{\log_{3/2} 1} = n^0 = 1$.

- $T(n) = T(2n/3) + 1$
- $a = 1$, $b = 3/2$, $f(n) = 1$.
- $n^{\log_{3/2} 1} = n^0 = 1$.
- $f(n)$ and n^0 grow at the same rate.

- $T(n) = T(2n/3) + 1$
- $a = 1,\ b = 3/2,\ f(n) = 1.$
- $n^{\log_{3/2} 1} = n^0 = 1.$
- $f(n)$ and n^0 grow at the same rate.
- Since $f(n) = \Theta(n^{\log_b a})$, case 2 of the master theorem gives us:

- $T(n) = T(2n/3) + 1$
- $a = 1,\ b = 3/2,\ f(n) = 1.$
- $n^{\log_{3/2} 1} = n^0 = 1.$
- $f(n)$ and n^0 grow at the same rate.
- Since $f(n) = \Theta(n^{\log_b a})$, case 2 of the master theorem gives us:
- $T(n) = \Theta(n^0 \lg n) = \Theta(\lg n)$

- $T(n) = 3T(n/4) + n \lg n$

- $T(n) = 3T(n/4) + n \lg n$
- $a = 3,\ b = 4,\ f(n) = n \lg n.$

- $T(n) = 3T(n/4) + n \lg n$
- $a = 3,\ b = 4,\ f(n) = n \lg n$.
- $n^{\log_b a} = n^{\log_4 3}$, where $\log_4 3 < 1$.

- $T(n) = 3T(n/4) + n \lg n$
- $a = 3,\ b = 4,\ f(n) = n \lg n$.
- $n^{\log_b a} = n^{\log_4 3}$, where $\log_4 3 < 1$.
- $f(n)$ grows faster than $n^{\log_4 3}$.

- $T(n) = 3T(n/4) + n \lg n$
- $a = 3,\ b = 4,\ f(n) = n \lg n$.
- $n^{\log_b a} = n^{\log_4 3}$, where $\log_4 3 < 1$.
- $f(n)$ grows faster than $n^{\log_4 3}$.
- So much faster that $f(n) = \Omega(n^{\log_4 3 + \epsilon})$, where $\epsilon = 0.2$.
- Can we apply case 2? If we prove the regularity condition!

- $T(n) = 3T(n/4) + n \lg n$
- $a = 3,\ b = 4,\ f(n) = n \lg n.$
- $n^{\log_b a} = n^{\log_4 3}$, where $\log_4 3 < 1$.
- $f(n)$ grows faster than $n^{\log_4 3}$.
- So much faster that $f(n) = \Omega(n^{\log_4 3 + \epsilon})$, where $\epsilon = 0.2$.
- Can we apply case 2? If we prove the regularity condition!
- For sufficiently large n, $af(\frac{n}{b}) = 3f(\frac{n}{4}) = \frac{3}{4} n \lg \frac{n}{4} \leq cf(n)$ because we get to choose $c = 3/4$.

- $T(n) = 3T(n/4) + n \lg n$
- $a = 3,\ b = 4,\ f(n) = n \lg n$.
- $n^{\log_b a} = n^{\log_4 3}$, where $\log_4 3 < 1$.
- $f(n)$ grows faster than $n^{\log_4 3}$.
- So much faster that $f(n) = \Omega(n^{\log_4 3+\epsilon})$, where $\epsilon = 0.2$.
- Can we apply case 2? If we prove the regularity condition!
- For sufficiently large n, $af(\frac{n}{b}) = 3f(\frac{n}{4}) = \frac{3}{4} n \lg \frac{n}{4} \leq cf(n)$ because we get to choose $c = 3/4$.
- By case 3 of the Master method, $T(n) = \Theta(n \lg n)$.